Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Cloud White: Detecting and Estimating QoS Degradation of Latency-Critical Workloads in the Public Cloud

Lucía Pons [a,*], Josué Feliu [b], Julio Sahuquillo [a], María E. Gómez [a], Salvador Petit [a], Julio Pons [a], Chaoyi Huang [c]

[a] Universitat Politècnica de València, Valencia, Spain
[b] Universidad de Murcia, Murcia, Spain
[c] Huawei Technologies CO., LDT., China

ARTICLE INFO

Article history:
Received 22 January 2022
Received in revised form 29 May 2022
Accepted 12 August 2022
Available online 17 August 2022

Keywords:
Cloud computing
Public cloud
Virtualization
Interference
Performance estimation
QoS
Tail latency
Latency-critical workloads

ABSTRACT

The increasing popularity of cloud computing has forced cloud providers to build economies of scale to meet the growing demand. Nowadays, data-centers include thousands of physical machines, each hosting many virtual machines (VMs), which share the main system resources, causing interference that can significantly impact on performance. Frequently, these data-centers run latency-critical workloads, whose performance is determined by tail latency, which is very sensitive to the interference of co-running workloads. To prevent QoS violations, cloud providers adopt overprovisioning strategies but they reduce the server utilization and increase the costs. A mechanism that accurately estimates performance degradation dynamically in a production system would allow cloud providers to improve the servers' utilization. In this work we propose Cloud White, an approach that is able to detect the inter-VM interference in scenarios with multiple co-located latency-critical VMs and estimate the performance degradation using multi-variable regression models. Unlike previous proposals, Cloud White is built taking into account the limitations of a public cloud production system. Experimental results show that Cloud White is able to estimate performance degradation with a small overall prediction error of 5%.

1. Introduction

An increasing amount of computing is being performed in public clouds, such as Amazon's EC2 [1], Microsoft Azure [2] and Google Compute Engine [3]. Cloud platforms provide two major advantages for end-users and cloud operators: flexibility and cost efficiency [4]. Users can quickly launch jobs without the cost of upgrading its computer infrastructure.

Cloud providers can achieve economies of scale by building large-scale datacenters and sharing their resources among different jobs. Following the typical virtualization model, the cloud provider allocates multiple virtual machines (VMs) in the same physical machine which provides fault isolation, security and improved manageability. However, sharing the physical machine means that inter-VM performance interference will appear due to multiple VMs competing for the major system resources (e.g., processor cores, last level cache (LLC), or main memory), making performance unpredictable. In other words, the interference adversely impacts on the quality of service (QoS) of the applications. Moreover, when the QoS degrades, it might not comply with the service level objectives (SLOs), stated in the service level agreement (SLA) between a cloud provider and a customer.

To avoid QoS violations and tackle the inter-VM performance interference, cloud providers typically adopt an *overprovisioning* strategy. That is, resources are assigned to each VM in excess to avoid possible performance degradation due to the inter-VM interference. This workaround, however, results in a poor utilization of the major resources of the cloud system. For instance, the average CPU utilization is typically far below 50% in cloud servers running latency-critical applications [5] and, in most cases, below 20% [6,7].

Despite the public cloud can run any kind of workload, an important characteristic that makes public cloud servers different from traditional computing nodes is that they frequently run latency-critical workloads, which is the main focus of this work. Most online or interactive services are examples of these workloads. Unlike conventional workloads, the performance of latency-critical applications is given by the obtained tail latency, indicated as a percentile (e.g., 95th or 99th) of all the latencies,

* Corresponding author.
E-mail addresses: lupones@disca.upv.es (L. Pons), josue.f.p@um.es (J. Feliu), jsahuqui@disca.upv.es (J. Sahuquillo), megomez@disca.upv.es (M.E. Gómez), spetit@disca.upv.es (S. Petit), jpons@disca.upv.es (J. Pons), joehuang@huawei.com (C. Huang).

https://doi.org/10.1016/j.future.2022.08.012

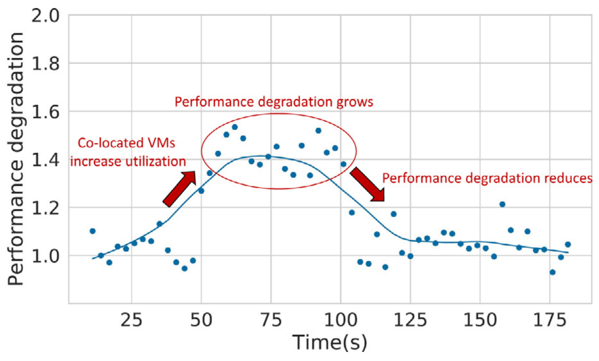


Fig. 1. Typical scenario: (1) co-located VM load grows and increases interference, (2) performance degradation increases for the target VM, (3) co-located VM reduces its load.

and accounts for the requests that take longer to complete. This means that the performance of these workloads is very sensitive to the inter-VM interference. Because of this fact, system resources need to be conservatively overprovisioned to ensure compliance with the SLA.

The previous rationale means that an important concern for cloud providers is to reduce the overprovisioning costs of the system resources associated to the VM; in other words, there is an interest in improving the resource efficiency to make the VM cost cheaper. For this purpose, in general, it would help that cloud providers could estimate online how performance of a given VM degrades depending on the co-located VMs on the same physical machine. In this situation, a VM can experience performance losses due to a change either in its load (*intra-VM*) or in the load of other VMs (*inter-VM*). Detecting the cause of interference inducing the performance loss is challenging. The former case refers to intra-VM interference and thus, it is not a concern for the cloud provider, which is the focus of this work. This paper concentrates on the latter case, where two or more VMs are involved and the performance degrades due to the inter-VM interference. We use the terms *victim* and *inflicting* to refer to VM(s) suffering performance degradation and the VM(s) increasing its load, respectively. Notice that a given VM can become a victim or act as inflicting across different periods of its execution time. The main goal of this paper is twofold: detect the victim VM(s) and estimate its performance degradation, both especially challenging when all the co-running applications are latency-critical in the public cloud.

We illustrate the problem in Fig. 1, which shows how the tail latency of a *victim* VM co-located with multiple resource-consuming VMs degrades over a 180-s time interval. The example starts in a steady situation with all co-located VMs running at a relatively low utilization (i.e., shared resources are not stressed). This steady situation is assumed to be the *normal conditions* or baseline situation, thus we assume that at this point of time the performance of the victim VM is not (or slightly) suffering due to the inter-VM interference (1 means no performance degradation). At second 50, one of the co-located VMs (*inflicting* VM) increases its load and starts making a higher consumption of the shared resources. This load increase impacts on the performance of other VMs. In the example, the victim degrades up to 1.4× – 1.6× its performance compared to the steady situation. After second 100, the resource consumption of the inflicting VM reduces and the performance degradation of the victim VM returns to a low level.

Discerning which are the victim and inflicting VMs is challenging in the public cloud, especially when running multiple latency-critical applications, since (i) VMs are seen as "black boxes" so the cloud provider has no information about the applications running on the VMs, and (ii) these applications present rather low shared resources utilization. Some previous works have tried to address these challenges. Unfortunately, most of them [8–13] fail in managing VMs as "black boxes" which makes them not applicable to the public cloud, or in dealing with latency-critical applications [14,15] (see Section 2 for further details).

This paper proposes Cloud White, an approach that not only identifies the victim VM(s) in cloud systems running multiple latency-critical applications as "black boxes", but also provides accurate estimates of their performance degradation. With our approach, the VM *is not* a black box anymore, but its behavior (and introduced interference) is revealed, becoming "white" or Cloud White. Cloud White uses the Giga Instructions Per Second (GIPS) to identify the victim VM(s). The novelty does not lie on the GIPS metric itself but we show its effectiveness for this purpose; in other words, as far as we know, it is the first time it is used to discern the victim VM(s). Performance degradation is estimated through multi-variable regression models.

Cloud White is, to the best of our knowledge, the first approach that is able to deal with the two aforementioned challenges: identify the victim VM(s) in cloud systems with latency-critical applications, and provide accurate estimates of its performance degradation. Moreover, the dynamic prediction error is, on average, below 10%.

In summary, this paper makes four main contributions:

- We propose a method to discern the victim from the inflicting VM(s) at execution time. More precisely, we propose using the Giga Instructions Per Second (GIPS) metric to identify the victim VM(s).
- We propose multi-variable regression models to estimate the 95th tail latency degradation due to interference caused by co-running latency-critical VMs. We prove that specific models for distinct processor utilization levels are needed to match the applications' performance degradation trend.
- We propose Cloud White, an approach that introduces the two previous mechanisms to make accurate estimates of the QoS degradation dynamically in a production system.
- The proposed approach works well when all the running applications are latency-critical and make smooth changes in the shared resource utilization.

2. Related work

Interference detection based on simple measurements. There is an extensive literature on detecting interference due to co-located jobs or resource sharing. Many approaches [17–20], however, monitor QoS (e.g., in terms of average latency or queries per second), something that cloud providers cannot carry out in real production-systems since VMs should be handled as black boxes.

Recent approaches [14,16] have tried to detect the inter-VM interference while complying with public cloud limitations. In [16], Javadi et al. propose Scavenger, a resource manager that considers tenant workloads as black boxes and identifies performance interference by monitoring the usage of a subset of the system resources (memory, network, LLC and CPU) consumed by the VM. Chen et al. propose Alita [14], which identifies contention online considering a different subset of the system resources (memory bus, LLC and power supply) based on low-level metrics but omitting important shared resources like disk and network. These approaches present two main shortcomings. On the one hand, performance interference is detected in a subset of the system resources. On the other hand, no prediction is made on how performance interference impacts on the QoS of VMs. Table 1 summarizes the main features of these approaches.

Performance interference prediction models. Prior works have proposed models to estimate the impact on performance caused

Table 1
Summary of closely-related work to Cloud White supporting latency-critical applications, sorted by chronological order.

Approach	Year	Main features				
		Interference detection	Performance interference prediction	Tail latency	All main system resources?	Black box?
DeepDive [13]	2013	✓, low-level metrics	✓, but in isolation	✗, average	✓	✓
RC [6]	2017	✗	✓, VM information	✓, median/99th	✗, no disk & network	✓
Scavenger [16]	2018	✓, VM resource usage	✗	✓, 90th/95th	✗, no disk	✓
Alita [14]	2020	✓, low-level metrics	✗	✗, average	✗, no disk & network	✓
Twig [11]	2020	✗	✓, low-level metrics	✓, 99th	✗, no disk & network	✗
CLITE [12]	2020	✗	✓, resource shares	✓, 95th	✓	✗
Cloud White	2022	✓, GIPS metric	✓, low-level metrics	✓, 95th	✓	✓

by the interference (e.g., introduced by co-runners or limited resources) in distinct environments or from different perspectives. In [21], online prediction models are built from a user-level perspective. This way allows user to run microbenchmarks to estimate resource contention at the shared resources and use application-specific models to estimate its impact on performance. Instead our work focuses on the cloud provider's point of view and therefore we assume no knowledge of the application running within each VM. Another important piece of research [8–10,13] has focused on predicting the performance interference that background applications introduce when co-located with latency-critical applications. Other works like [15] focus on co-location of HPC applications. From these works, [8–10,15] access data regarding performance of individual applications not accessible by cloud providers, which makes these approaches not practical in real production-systems. DeepDive [13] detects performance interference based on the aggregated resource system utilization of VMs by monitoring low-level metrics. Nevertheless, an important weakness of DeepDive is the *isolation-based methodology* used to quantify interference in VMs, which is prohibitive in public cloud environments. Resource Central [6], based on that certain VMs show consistent behaviors over multiple lifetimes, learns off-line from past behaviors to predict online future behavior of customer's VM. This approach predicts high-level metrics like CPU utilization, cores, memory or workload class. To this end, it uses machine learning methods (e.g., random forest) to output buckets, instead of predicting a single number through regression models. Some approaches [11,12] have used prediction models to estimate the resulting performance when modifying a target resource configuration. Twig [11] uses performance counters to estimate the QoS that results from different DVFS and core combinations. CLITE [12] elaborates prediction models to search for the near-optimal resource partitioning. However, both Twig and CLITE require monitoring at run-time the performance (e.g., latency) of the executing applications, thus not considering VMs as black boxes. In addition, performance degradation due to interference is continuously being estimated, instead of being estimated only when it is detected.

3. Experimental setup

3.1. Platform

With the aim of building a controlled environment closely resembling one of the public cloud, we set up an experimental platform following the structure depicted in Fig. 2. The three main components of the system are the physical server (main node), the client node, and the storage node, interconnected among them with two 20 Gbps dedicated links.

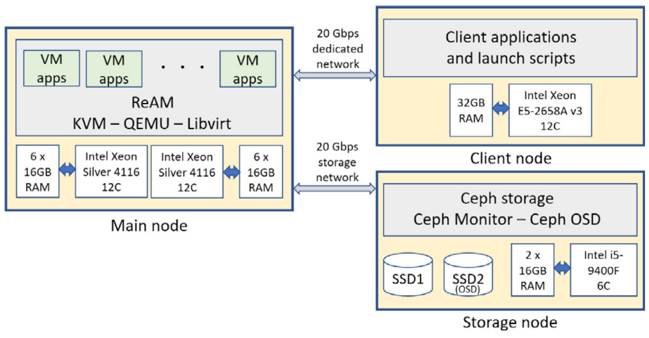


Fig. 2. Overview of the complete experimental framework.

The server node is made up of two 12-core Intel Xeon Silver 4116 processors, with six memory channels holding 16 GB each, which amounts 96 GB (6 × 16 GB) of DRAM and supports a bandwidth of 107.3 GB/s. It has a 16.5 MB 11-way LLC, which supports Intel Resource Director Technologies (RDT) [22]. The installed operating system is Ubuntu 18.04 LTS. The main tasks carried out in the server node, performed under our Resource and Application Manager software (ReAM), are (1) run the VMs hosting the applications to be studied, (2) collect the data sampled, and (3) apply Cloud White. Data sampled include (i) hardware/software events gathered with the performance monitoring unit (PMU) using Perf [23,24], (ii) breakdown of the processor time from the `/proc/stat` system file, which reports statistics about the amount of time spent performing different kinds of work, and (iii) consumption of the main system shared resources not monitored with the PMU but with the support libraries offered by the developers of the tools. LLC occupancy and memory bandwidth are measured with Intel's PQoS library [25]. Disk bandwidth is obtained with libvirt's library [26]. Network bandwidth (both send and receive) is obtained with OVS-ofctl [27] commands, used to monitor and manage OpenFlow switches.

To replicate a typical cloud infrastructure, the experimental framework works with VMs that use KVM as hypervisor, QEMU as virtualizer, and libvirt as virtualization manager. This configuration is similar to that used by a compute node of OpenStack [28], which is currently used in production-system software [29]. Although OpenStack supports many hypervisors, KVM is the most widely adopted hypervisor in the OpenStack community [30]. To connect the physical network interfaces with the VMs, the main node has a virtual switch configured using Open vSwitch [27] (OvS) and Data Plane Development Kit [31] (DPDK). OvS and DPDK enable direct transfer of packets between the user space and physical interface, bypassing the kernel network stack, and therefore, boosting network performance over the default packet forwarding mechanism. The server node accesses a remote storage system implemented with Ceph [32], one of the preferred block storage backends in public clouds using OpenStack. The client node has an Intel Xeon E5-2658 A v3 processor with 12 cores. It is only used to run the client of each workload, which generates and sends requests to the server.

In summary, our experimental platform includes all the components of a typical cloud system [33]. Even though it is not composed of many nodes like a production system, the setup of one server node and one client node complies with the twofold objective of this work: (i) study the interference among VMs executing in the same server node, and (ii) use the client node to carry out experiments with controlled load that help us evaluate (even saturate the server) and validate our approach.

3.2. Workloads

In this work we use a set of representative latency-critical benchmarks from the Tailbench suite [34]: ***img-dnn***, ***masstree***, ***moses***, ***silo*** and ***specjbb***. Client requests are issued to the server following the Zipfian distribution, which accurately models request times in online services [35,36]. According to the characteristics of these applications, our results confirm that they are mainly limited by CPU or memory resources [37], thus, two different models will be required to predict the performance (see Section 6). We extended Tailbench source code to report tail latency of the requests serviced dynamically after a configurable time slice, set to 1 s in our experiments.

3.3. System load conditions

A key modeling decision is to establish the load level of interest to be studied. The client load is controlled by varying the queries per second (QPS) performed by clients. However, the QPS is an internal metric of the workload unknown to the cloud provider. Thus, we consider the CPU utilization as a proxy of the load from the cloud provider perspective.

To this end, two main CPU utilization levels are of interest for the cloud provider: normal and overloaded conditions. Normal conditions refer to low processor utilization (e.g., 10%), where the interference introduced due to the VM load is negligible. Current cloud servers [6,7] typically run at this low CPU utilization. Overloaded conditions refer to a relatively high processor utilization (e.g.,over 50%), where cloud providers assume that the system SLO may be violated. It is important to note that the utilization value strongly depends on the type of workload. For instance, if performance is measured in terms of tail latency, small CPU variations can turn into a large tail latency increase and cause important performance degradation [37]. We considered CPU utilizations ranging from 10% to 60% to analyze the impact of varying the inflicting VM's load on the performance of the victim.

4. Cloud White: Detecting the inter-VM interference

4.1. GIPS: a new way to detect the victim and inflicting VMs

One could think that the CPU utilization could be used as an intuitive solution to detect the VM causing the interference. The CPU utilization, however, does not only grow when the QPS of the target VM grows during its execution, but also due to the inter-VM interference. The reason is that the CPU utilization comprises both the time the processor is executing instructions, which grows with the QPS, and the time it is stalled waiting for memory accesses, which grows with interference. This means that, apart from the CPU utilization, other metrics are needed, as discussed below.

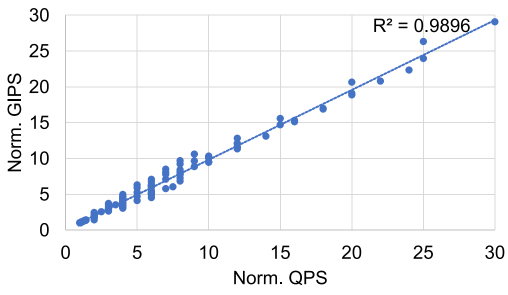


Fig. 3. Relationship between GIPS and QPS for the five studied latency-critical applications.

To cope with this challenge, a wide set of performance metrics (like the IPC, ROB stalls, etc.) were analyzed. We found that the number of instructions executed per second helps identifying the inflicting and victim VMs. More precisely, the **Giga Instructions executed Per Second** (GIPS). Based on this finding, the devised approach uses the GIPS to identify the victim VM(s). Notice that the novelty is not the metric itself but the purpose for which it is used. The rationale is as follows. The amount of instructions committed by a VM per second is directly related to the number of queries that it receives per second (QPS). That is, if a victim VM keeps its load steady (i.e., the server processes the same requests per second), its GIPS remain unaffected. Otherwise, the GIPS will change.

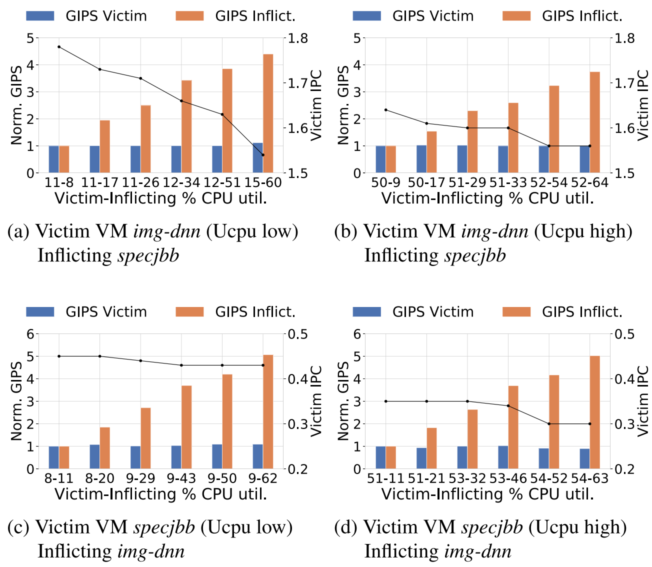
To support this claim, we studied the correlation between QPS and GIPS. Fig. 3 plots the results, varying the QPS for the five studied applications executed under four server threads (1, 2, 4 and 8) configurations. Tested QPS ranges from 100 to 2000 (i.e., the maximum value before the 95th tail latency saturates). For each server configuration, both QPS and GIPS, have been normalized to that of the lowest QPS value (i.e., 100). A total of 175 points are plotted. Results show that the GIPS presents a very strong linear correlation ($R^2 = 0.9896$), almost perfect, to QPS.

It should be taken into account that cloud workloads do not stress 100% the CPU, but the CPU utilization usually takes low values (e.g., 20%) [6,7]. Therefore, IPC and GIPS are not directly correlated since IPC is calculated over the time the CPU is busy (i.e., utilization time) while the GIPS is computed over the total measurement time, which comprises both the utilization time and time the VM is waiting for client requests.

4.2. Case study with two VMs

This section shows how the GIPS helps detecting the victim and the inflicting VMs running latency-critical applications through an experimental example. For illustrative purposes, only two VMs are used, one acting as the victim and another as the inflicting.

Fig. 4 shows the GIPS and the IPC varying the CPU utilization of the victim and inflicting VMs across four different scenarios. In each plot, each pair of bars corresponds to a different experiment. The X-axis legend below each bar indicates the CPU utilization that the victim and inflicting VMs achieve in the experiment. Two load (utilization) levels, low (left-side plots) and high (right-side plots), are considered for the victim VM, which refer to about 10% and 50% CPU utilization in isolation, respectively. The utilization of the inflicting VM has been set to grow from about 10% (no contention) up to 60%. More precisely, QPS have been experimentally configured in order to achieve that utilization. The first pair of columns refers to a low contention scenario where



(a) Victim VM *img-dnn* (Ucpu low) Inflicting *specjbb*

(b) Victim VM *img-dnn* (Ucpu high) Inflicting *specjbb*

(c) Victim VM *specjbb* (Ucpu low) Inflicting *img-dnn*

(d) Victim VM *specjbb* (Ucpu high) Inflicting *img-dnn*

Fig. 4. Normalized GIPS of victim VM (fixed QPS) and inflicting VM (increasing QPS) for different CPU loads, high and low. Right *Y*-axis shows the IPC obtained (black line) by the victim VM in each of the test cases.

the inflicting VM presents low load and it is used as baseline. The GIPS of both VMs are normalized to the achieved in the baseline scenario.

In the upper plots, *img-dnn* is the victim VM under low load (Fig. 4(a)) and high load (Fig. 4(b)), and *specjbb* is the inflicting VM growing the interference from left to right. This can be appreciated, as the normalized GIPS of the inflicting VM (orange bar) grows in a by 4× factor in the highest interference. On the contrary, the GIPS of the victim VM (blue bar) remain around 1 across the six experiments in the plot. The interference introduces an important change in the victim behavior. The CPU utilization grows from 11% to 15% when it works under low load (left side plot) and from 50% to 52% (right side plot). This represents an important growth (in percentage), especially under low load conditions, thus it translates into significant performance losses. Notice that the IPC of the victim VM (right Y-axis) significantly decreases as the interference increases, dropping from around 1.78 to 1.53.

In the bottom plots (Figs. 4(c) and 4(d)) the applications change their role: *img-dnn* is the inflicting VM and *specjbb* is victim VM. Similar results can be observed. Again, the GIPS of the victim VM are not affected as the interference increases while the GIPS of the inflicting VM grow up in a 5× factor. Unlike the previous figures, the CPU utilization of the victim grows in a negligible way (less than 1%); consequently, its IPC drops to a lesser extend, especially in the low load plot (Fig. 4(c)). The reason is that *img-dnn* is more memory-intensive than *specjbb*, and thus, is more sensitive to the co-runner's workload.

Finally, we would like to remark that the GIPS metric is not limited to two VMs but it can be applied regardless of the number of VMs (see Section 7.3).

4.3. Cloud White: Interference detection phase algorithm

Algorithm 1 presents the pseudo-code of the interference detection mechanism performed by Cloud White. That is, the steps carried out to discern if a given VM present an inflicting or a victim behavior. First, the normalized load and GIPS are computed for each application. Load is quantified in terms of percentage of *guest time* (from the `/proc/stat` system file, see Section 6.1).

Algorithm 1 Cloud White's interference detection phase pseudo-code where inflicting and victim behaviors are detected.

```
1:  for all apps do
2:    Compute $\%guest_{norm}$ and $GIPS_{norm}$
3:    if $\%guest_{norm}$ increases M% then
4:      if $GIPS_{norm}$ increases N% then
5:        Inflicting behavior
6:        Tag as victim all non-inflicting VMs
7:      else
8:        Victim behavior
9:      end if
10:   else
11:     VM's behavior is steady
12:   end if
13: end for
```

Normalized values are computed with respect to *No Interference* scenario (see Section 7.1 for more details). Once computed, Cloud White checks if the VM has experienced a load change. This is done by checking if *%guest time* has changed – increase or decrease – more than a given threshold (*M*%) over the rolling mean of the previous intervals. If this condition fulfills, then it checks if the GIPS have also noticeably changed (over *N*%). In such a case, the VM is tagged as inflicting. Otherwise, it is tagged as victim. Notice that some VMs may not experience any load change (lines 10–11). However, if one of the co-located VM(s) exhibits an inflicting behavior, the VMs exhibiting a steady behavior are also tagged as victim as they may be possibly affected in the near-future by the interference caused by the inflicting VM(s). The results presented in this work have been obtained with *M*% and *N*% set to 5 and 50, respectively.

5. Microbenchmarking: Introducing inter-VM interference

To devise the models to estimate the performance degradation, we first need to study the impact of the interference on the system performance. This study needs to be done in a controlled way, that is, we need a method to control the introduced interference. A practical option commonly employed is the use of microbenchmarks, or synthetic benchmarks, especially designed to stress specific systems resources (e.g., the cache, the main memory or the disk) by introducing interference in their utilization. One of the main advantages of using microbenchmarks is that they can be finely tuned, setting different utilization levels on each shared resource. To generate a workload that introduces realistic interference, we have used the *stress-ng* [38] microbenchmark, which stresses the three main system components (LLC, main memory bandwidth and disk). In addition, we used *iPerf3* [39] to stress the network bandwidth.

In the experiments, all (24 logical) cores of our machine are used to model scenarios closely resembling to a production system. Two different instances of the microbenchmark are used. One instance is devoted to stress one or some shared resources, and is hosted in a VM with 8 logical cores. The other instance, referred to as *no interference* has been used to introduce relatively low interference, that will barely affect the victim VM. To this end, 6 logical cores are occupied presenting each of them a low utilization. In this way, the stress introduced comes only from a subset (i.e., 8) of the cores of the system as it is likely to occur in real systems. From the remaining logical cores, 2 were devoted to the victim VM and 8 to run the software agent components (OVS, DPDK, and ReAM described in Section 3). From now on, we assume the same core distribution of the software agents.

Below, we present five types of workload scenarios created with the microbenchmarks to analyze different stressing levels at the major system components.

No interference. This scenario represents the common situation of the target system, where there is a low (by 10%) processor utilization and resource usage. It will be used as our baseline scenario to check performance degradation.

Cache-memory stressing. This scenario is used to study the effect of stressing the LLC occupancy and main memory bandwidth. Three main stressing levels have been considered, ranging memory bandwidth consumption from 1.9 to 5.5 GB/s and LLC space from approx. 12.5 to 14.7 MB.

Disk stressing. This scenario implies a high number of write and read operations to disk. Three stressing levels have been considered, from moderate (around 45 MB/s) to high (over 100 MB/s) disk bandwidth consumption.

Cache-memory-disk stressing. This scenario adds the stress introduced by the cache-memory and disk scenarios within the same experiment.

Network stressing. This scenario stresses the network bandwidth (both receive and transmit, but in separate experiments).

6. Cloud White: Modeling performance degradation

After using the microbenchmarks to model different levels of interference, we analyze the impact of the interference on tail latency. This analysis is aimed at identifying the metrics that have stronger relationship with performance degradation in order to consider them in the prediction models. Recall that the public cloud imposes important limitations for an approach to be practical. First, it should not have prior knowledge of the workloads running on the VMs. Second, it should not require costly actions (e.g., isolating a VM). And third, it should be general enough to adapt to different workloads and system conditions.

The devised models are generic and can be applied to other applications showing similar characteristics. To check this claim, we make two different groups of workloads: one for building the models (*known workloads*) and another for evaluating the models (*unknown workloads*), not previously used. From our set of applications, we identified two behaviors, CPU- and memory-bound (see Section 3.2). In case Cloud White encounters an application presenting an unseen behavior, models would be trained with this application to update or generate new models. We use *specjbb* and *img-dnn* as known workloads presenting CPU- and memory-bound behaviors, respectively, and leave *silo*, *masstree* and *moses* as unknown applications.

6.1. Looking for performance metrics as model parameters

Since the cloud provider sees tenant VMs as "black boxes", we studied a wide set of performance indicators from the main system components (processor, memory, network and disk) with the aim of finding potential correlation between them and performance degradation.

We evaluated two different CPU metrics related to where the processor spends time: *guest* time, which accounts for the time spent by processes running on a virtual CPU, and *idle* time, which represents the time spent idling (i.e., not executing instructions) while there are no disk I/O requests outstanding. To assess the core performance, we have evaluated metrics to quantify throughput (e.g., IPC) and the interference within the core (e.g., processor stalls and L1 cache misses). In addition, we have studied metrics that quantify the LLC occupancy and bandwidth consumption in the main memory, disk, and network. Regarding network bandwidth, we found out that the Tailbench applications consumption is very low compared to the maximum supported bandwidth (over 20 Gb/s) in our experimental platform. Thus, no performance degradation was observed due to the interference in the network bandwidth. Furthermore, the cloud provider should always provide enough network bandwidth to almost completely avoid the interference at this component because queuing delays at the network will immediately prevent the application from meeting the required QoS.

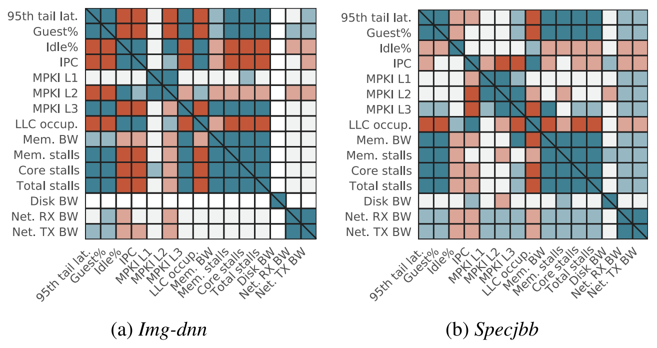


Fig. 5. Correlation of the studied metrics for *img-dnn* and *specjbb* for a 20% processor utilization.

Fig. 5 illustrates the correlation between the degradation of the 95th tail latency and the studied metrics (left-most column and top row) for the two applications representative of memory- and CPU-bound behaviors, *img-dnn* and *specjbb* running a QPS that results in an average CPU utilization of 20%. That is, how much performance degradation (compared to that obtained in the *No Interference* scenario) is related with each of the studied performance metrics. Additionally, the correlation between each pair of metrics is shown. Each table represents the correlation of the studied metrics for each application, obtained from the results of all the experiments performed with the microbenchmarks in all the stressing scenarios previously discussed. Positive correlations are colored in blue and negative correlations in red, where a darker shade implies stronger correlation. Correlations between 0.2 and 0.6 are colored in light blue and correlations between 0.6 and 1 in dark blue. Negative correlations within the same range are colored in light and dark red. Cells colored in white represent a very low correlation (i.e., between −0.2 and 0.2). As observed in the *95th tail lat.* column, *img-dnn's* performance achieves a strong correlation with the processor utilization, core and LLC metrics. However, *specjbb* achieves strongest correlation in just a subset of these metrics (percentage of guest time and processor stalls) and in the main memory bandwidth. Even though *specjbb* is CPU-bound, memory bandwidth has a stronger correlation with performance degradation in *specjbb* than in *img-dnn*. *Specjbb*'s tail latency is 4× shorter than that of *img-dnn* and consumes more memory resources, so a small impact in the memory bandwidth can turn into performance degradation [37]. Note, however, that the correlation factor indicates how related are the variations in both metrics but gives no insight about the impact on performance of the interference in that component. In both cases, network- and disk-related metrics achieve a poor correlation, being stronger in *specjbb* but not significant enough. This was expected because the two workloads present relatively low disk and network utilization.

6.2. Relationship between system load and the models

So far, we have considered that the CPU utilization of the victim remains constant and the CPU utilization of the inflicting ranges from 10% to 60% (higher utilization normally translates into higher interference) to analyze workload conditions of interest (see Section 3.3). However, in the real cloud, the processor utilization of the victim does not remain constant but it tends to vary in the range from normal to overloaded.

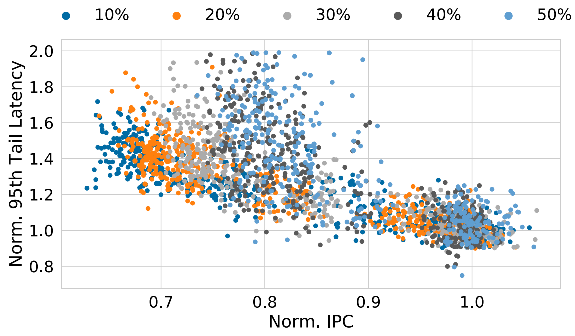


Fig. 6. Relationship between IPC and performance degradation for *img-dnn* varying the CPU utilization (10% to 50%).

Under different CPU utilizations, we observed that the trend of the performance metrics also varies. An example can be observed in Fig. 6, showing the relationship between dynamic performance degradation (95th tail latency) and IPC for the studied CPU utilization scenarios for *img-dnn*. Each point corresponds to the 95th tail latency and IPC achieved over a time interval of 5 s, resulting in more 1000 points plotted in the graph.

Values have been normalized to the average value obtained with the *No Interference* workload scenario (see Section 5). As observed, the slope of the points corresponding to each CPU utilization (with different colors) varies, being more pronounced for higher processor utilization. Additionally, points that belong to higher CPU loads (i.e., 40% and 50%) present a non-linear trend. Therefore, building a unique model that embraces all processor utilizations may lead to high prediction errors. To deal with this fact, specific models can be built for distinct processor utilization (e.g., 10%, 30% and 50%).

6.3. Multi-variable regression models

To acquire a sound knowledge about the performance-related metrics (see Section 6.1), we analyzed how each metric individually correlates with performance degradation using regression models. We found that none of them strongly correlate with performance in a generalized way. Therefore, we looked into multi-variable regression models to devise models that achieve a stronger correlation. These models pursue to improve the correlation by combining several metrics.

Designing statistical models. Due to the non-linear nature of the tail latency metric, linear regression models are not the most suitable models to use. We also observed that the performance degradation metric does not follow a normal distribution.

Fig. 7 shows the distribution of the performance degradation (95th tail latency normalized over *No interference*). A value of 1 in the *X*-axis means no performance degradation. Results were gathered in the stressing scenarios for *img-dnn* and *specjbb* for two processor utilization levels: 20% (upper plots) and 50% (bottom plots). As observed, the histograms do not exhibit a symmetric shape. As expected, in the top plots where the CPU utilization (20%) is lower, the performance is less affected by the interference. This can be appreciated in that, in around 96% of the observations, performance degrades by a factor of 1.9×. In contrast, in the highest CPU load scenario (Figs. 7(c) and 7(d)), this range is extended up to around 2.8, meaning that in some cases performance degradation can be as much as 180%.

In all cases, the histogram shape does not resemble a typical bell-shaped curve but is right skewed, especially in the highest CPU load plots, illustrating that the Gaussian distribution is not the most appropriate distribution for tail latency [40]. To

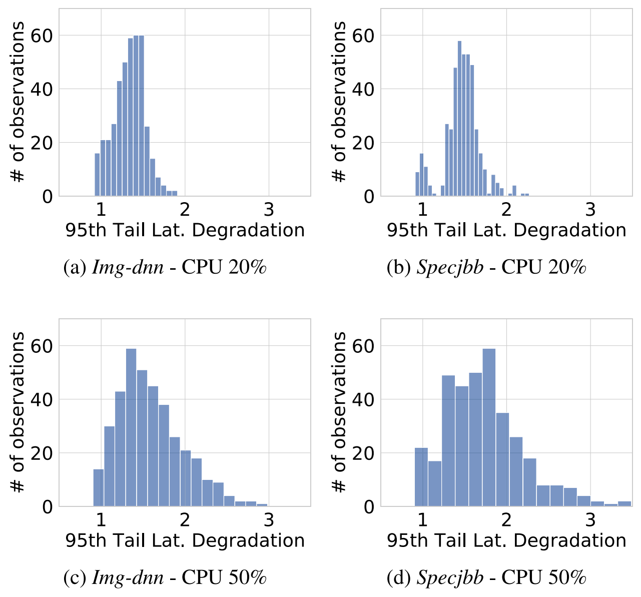


Fig. 7. Histograms with the frequency distribution of the 95th tail latency degradation values in different scenarios.

check this, we modeled the measured data linearly and performed the Anderson–Darling test [41] which confirmed that, generally, the models' residuals were not normally distributed. We found that only *img-dnn's* 50% CPU load model satisfied the normality assumption, so this can be considered as the rare case. Additionally, we checked that in most cases, performance degradation did not present a linear relationship with the studied metric (e.g., IPC values corresponding to 40% and 50% CPU utilization in Fig. 6). Therefore, for generalization purposes, we looked into other models that do not require the data to be normally distributed. Among these, we found that the generalized linear models (GLM) [42] are the most appropriate for non-linear and skewed distributions [43].

Model fitting - reduction strategy. The models were built and fitted using statsmodel Python library [41]. The model fitting was performed with the iteratively reweighted least squares (IRLS) method, with the objective of minimizing the deviation that occurs when estimating performance degradation. No constraints were specified to obtain the model coefficient values. A model reduction strategy [44] by statistical significance of the terms has been followed to find the truly significant variables. This implies that, initially, the model is built including all candidate variables (performance metrics), and the least statistically significant variable (largest P>|z|, fulfilling that is higher than 5%) is removed. This process is repeated until the model only contains significant terms. In this work, we considered candidate variables the performance metrics achieving a correlation higher than 0.4.

For illustrative purposes, Table 2 shows an example of the model generated with the data from the experiments performed with *specjbb* under stressing conditions (see Section 5) for 20% CPU utilization. The upper part of the table shows a summary of the characteristics of the resultant model, both qualitative and quantitative. In this example, the obtained model belongs to the GLM Gamma exponential family, with inverse_power (i.e., reciprocal) link function, the default link function for the Gamma family. Other link functions were explored like the logarithmic function, but the best results where obtained with the inverse_power. We would like to remark that the model characteristics are the same for the models generated for all the CPU utilization levels. Results like the `Log-Likelihood`, `Deviance` and `Pearson`

Table 2
Example of the parameters and statistics of the prediction model generated with data from *specjbb* with a 20% CPU load.

Generalized linear model regression results			
Dep. Variable:	perf. deg.	No. Observations:	425
Model:	GLM	Df Residuals:	420
Model family:	Gamma	Df Model:	4
Link function:	inverse_power	Scale:	0.0043907
Method:	IRLS	Log-Likelihood:	396.96
No. Iterations:	6	Deviance:	1.7979
Covariance type:	nonrobust	Pearson chi2:	1.84

	coef	std err	z	$P > \|z\|$	[0.025]	[0.975]
const	1.0602	0.017	63.261	0.000	1.027	1.093
guest%	−0.0207	0.009	−2.261	0.024	−0.039	−0.003
Mem. BW	−0.0233	0.002	−10.432	0.000	−0.028	−0.019
Core stalls	−0.1674	0.017	−9.713	0.000	−0.201	−0.134
Total stalls	0.0876	0.016	5.446	0.000	0.056	0.119

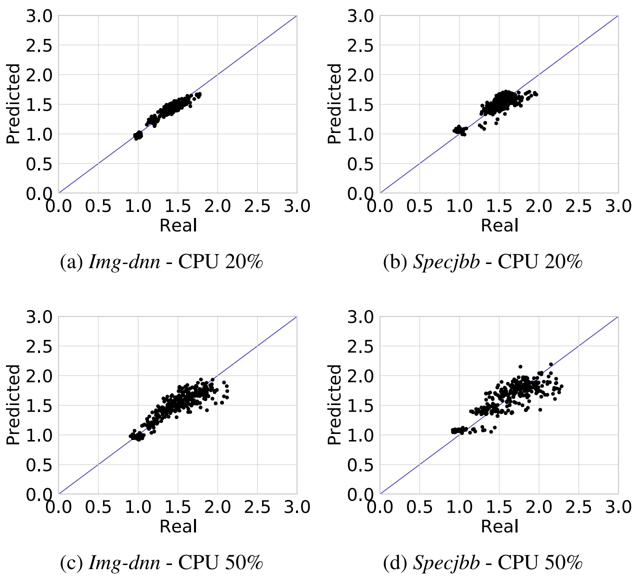


Fig. 8. Scatter plots with real (measured) vs. predicted performance degradation.

`chi2` [45] indicate the model goodness to the data used to train it. For instance, deviance is a measure of goodness of fit (the lower and closer to zero, the better).

The bottom part of Table 2 shows the coefficients of the model variables, as well as the standard error of each one. Among the generated models, this is the part that mainly differs, since each model has different (both in number and value) coefficients. As a result of the model reduction, all independent variables present a P>|z| lower than 5%. In this example, the model is made up of 5 independent terms, that consists of a constant term and 4 significant variables (degrees of freedom, Df, equals 4): guest%, memory BW, core stalls and total stalls.

The goodness of the model can also be analyzed graphically. Fig. 8 shows four examples of the relationship between the real performance degradation values with the predicted values for *img-dnn* and *specjbb* in two different CPU utilization scenarios. Results show that most of the points lie on the curve $x = y$ (plotted diagonal) or very close to it, which indicates the *perfect prediction* (i.e., real and predicted values are equal). Higher processor utilization scenarios (Figs. 8(c) and 8(d)) show higher spread, something expected since tail latency is more affected with higher CPU utilization (i.e., overloaded scenario).

Final models. This section presents the memory-bound and CPU-bound models devised in this work for *img-dnn* and *specjbb*, respectively. Eqs. (1) and (2) show the formulas where K, $x1$, $x2$ to xn represent the coefficients of the constant term and the independent variables, respectively. y_{mem} and y_{CPU} represents the dependent variables in terms of performance degradation for the memory-bound and CPU-bound models, respectively.

$$y_{mem} = \frac{1}{\begin{pmatrix} K + (x1 \times idle\%) + (x2 \times IPC) \\ + (x3 \times MPKI\_L2) + (x4 \times MPKI\_L3) \\ +(x5 \times mem\_BW) + (x6 \times mem\_stalls) \\ +(x7 \times core\_stalls) + (x8 \times total\_stalls) \end{pmatrix}} \quad (1)$$

$$y_{CPU} = \frac{1}{\begin{pmatrix} K + (x1 \times guest\%) + (x2 \times idle\%) + (x3 \times IPC) \\ +(x4 \times mem\_BW) + (x5 \times MPKI\_L2) \\ +(x6 \times MPKI\_L3) + (x7 \times mem\_stalls) \\ +(x8 \times core\_stalls) + (x9 \times total\_stalls) \end{pmatrix}} \quad (2)$$

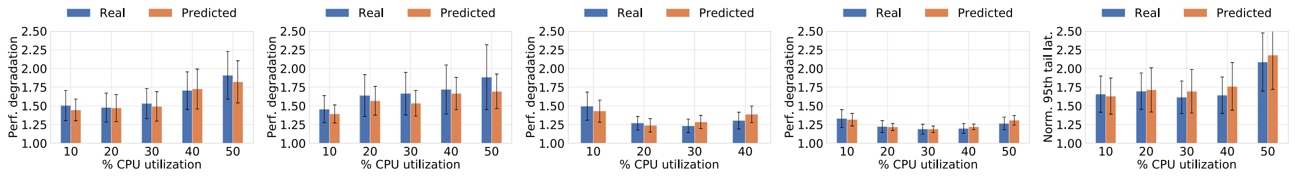
Since performance degradation strongly depends on the CPU utilization, each model needs to be tuned for five CPU values (i.e., 10%, 20%, 30%, 40% and 50%). We found that this number is enough to provide accurate estimates. For each individual model (i.e., model generated for a CPU load), only those statistically significant terms have been considered, meaning the coefficients of the remaining terms are set to zero. For instance, in the model described in Table 2 only the variables of the guest%, memory bandwidth, core stalls and total stalls metrics are significant and therefore, their parameters are not equal to zero.

7. Cloud White evaluation

7.1. Methodology

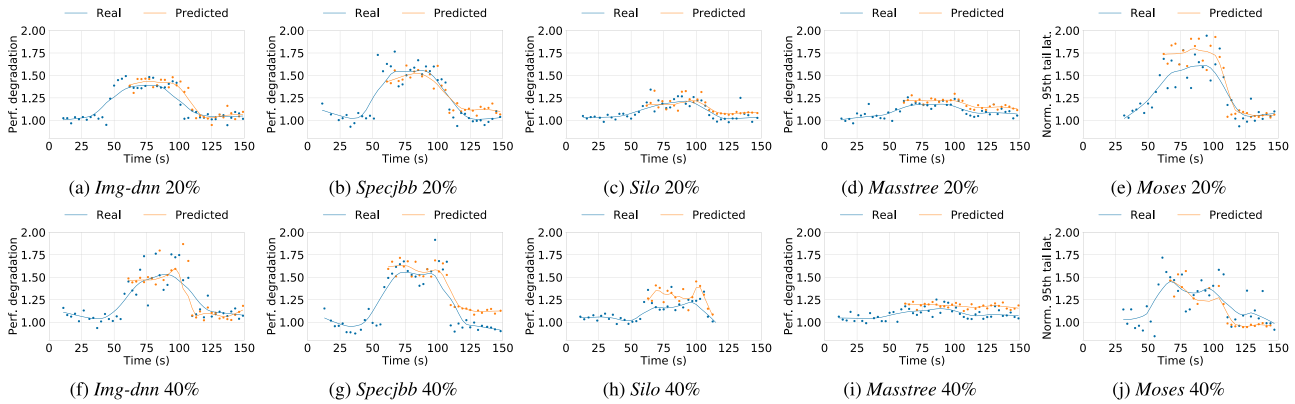
To evaluate the model accuracy, we performed a wide set of experiments both with one and multiple latency-critical applications. Firstly, we tested the effectiveness of the models by executing the target latency-critical application in a VM (*victim VM*) with two virtual CPUs (VCPUs) along with a stressor microbenchmark, launched in another VM (*inflicting VM*) running on eight VCPUs. Then, experiments were performed with multiple (two and three) latency-critical applications running in different VMs to test if Cloud White could effectively distinguish the victim and inflicting VMs and estimate the performance degradation of the victim VM. Some applications were configured to increase their load (i.e., QPS) gradually at different points of the execution time to dynamically introduce or remove interference. Each VM was launched with 2 (victim) or 4 (inflicting) VCPUs. The remaining CPUs were occupied by an instance of the microbenchmark running the *No Interference* scenario launched in a VM with 8 VCPUs.

For the experiments performed with the microbenchmark, we configured it to run the *No Interference* scenario for the first M seconds. During this time, Cloud White sees that the system presents a low and constant load and gathers the baseline values. Remember that the public cloud runs, most of the time, in steady phases with low CPU loads [5,7], therefore baseline values would be gathered during these times. After this time, the microbenchmark adopts the stressing cache-memory-disk model (see Section 5) for N seconds. When this happens, Cloud White detects that a load change is taking place in the VM running the microbenchmark (i.e., acts as inflicting) and starts to apply the regression model to the VM identified as victim to estimate its performance degradation. Finally, the microbenchmark adopts again the *No Interference* model in order to check if Cloud White is able to detect this new situation and estimate that little or no performance degradation is taking place. In this work, M and N are set to 60 s and 50 s, respectively. In a similar way, in the multiple latency-critical applications experiments, we have



(a) *Img-dnn* (b) *Specjbb* (c) *Silo* (d) *Masstree* (e) *Moses*

Fig. 9. Bar plots comparing the overall real and predicted performance degradation (i.e., 95th tail latency). The CPU utilization refers to that achieved, on average, when applications were executed with *No Interference*.



(a) *Img-dnn* 20% (b) *Specjbb* 20% (c) *Silo* 20% (d) *Masstree* 20% (e) *Moses* 20%

(f) *Img-dnn* 40% (g) *Specjbb* 40% (h) *Silo* 40% (i) *Masstree* 40% (j) *Moses* 40%

Fig. 10. Dynamic graphs comparing the real and predicted performance degradation (i.e., 95th tail latency).

configured inflicting applications to start increasing the load after a minimum of 45 s so Cloud White can gather the baseline values during this time.

Experiments were repeated five times, point at which the deviation among the measured 95th tail latencies was less than 5%, except for some experiments with higher CPU load (e.g., 40% and 50%) that experience higher deviation, between 7% and 10%. Prediction accuracy results were evaluated in terms of (1) *overall prediction error*, which defines the difference between the real and estimated performance degradation of the entire experiment, and (2) *average dynamic prediction error*, which quantifies the average error of each prediction performed with respect to the real value in each 1s quantum. In both cases, performance degradation quantifies the 95th tail latency normalized against that obtained in a phase under low interference (e.g., first M seconds of the execution). Therefore, a value of 1.0 means no performance degradation, and higher values mean degradation has taken place.

7.2. Prediction effectiveness with a single latency-critical application

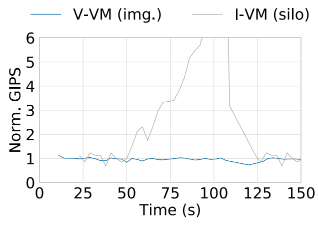
This section shows the results obtained when applying Cloud White to the set of studied applications executed individually together with the microbenchmark. This includes those applications left outside for validation purposes (*masstree*, *moses* and *silo*), as well as those used to create and train the models (*img-dnn* and *specjbb*). The goal of including the latter two applications in the evaluation is to check if Cloud White correctly applies the appropriate model when predicting performance degradation. Notice that, in all cases, Cloud White sees these applications as black boxes and has no prior information of the applications under execution, other than the resource consumption and hardware/software events gathered at run-time with the PMU.

Overall results can be observed in Fig. 9.[1] Each graph presents the normalized real or measured performance degradation (blue bar) and the normalized estimated performance degradation (orange bar) for a variety of CPU utilization levels (X-axis), calculated as the 95th percentile of all the measurements and estimations made at run-time, respectively. Interval bars show the deviation achieved for the run-time values of both the measured and estimated normalized latency in the experiments. As it can be observed, Cloud White is able to estimate performance degradation accurately for most of the experiments, having on average a 5% prediction error. Experiments with high CPU load (i.e., 40% and 50%), however, make slightly higher prediction errors (7.5%). This is mainly due to the fact that applications are close to saturating and, therefore, 95th tail latency shows more spike values. This is also reflected in the interval bars, as longer interval bars mean that there exists higher deviation among values. In general, we can observe that the estimated values' deviation is similar to the real values' deviation.

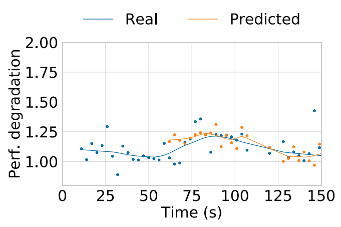
A complete evaluation, however, should also consider how Cloud White behaves when the workload changes dynamically. Fig. 10 shows examples of how real (measured) performance degradation evolves across the execution time, and the predicted performance degradation performed by the models for experiments with 20% and 40% CPU load for the first 150 s of execution.[2] Since 95th tail latency metric experiences high variation, a best fit curve computer with the LOWESS algorithm [46] has been plotted instead of line joining all the points. Notice that prediction starts around second 60, point at which Cloud White detects that the co-running VM is changing its load and it is possibly causing interference. As it can be observed, both the real and predicted curves follow a similar trend in most time intervals, showing Cloud White is able to detect and predict accurately performance degradation. On average, dynamic prediction error is less than 10%, except for some experiments like in *specjbb* with 40% CPU utilization which is higher. Likewise, *moses* experiences a higher sensitivity to interference at the shared resources obtaining higher prediction errors in some experiments, but in all cases, average prediction error is less than 20%.

[1] Silo shows no results for 50% CPU utilization since the application saturates before reaching this load.

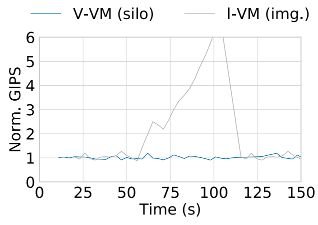
[2] *Silo* 40% has a shorter execution time since higher time resulted in QoS violation.



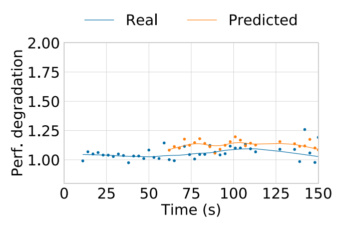
(a) GIPS victim VM *img-dnn*



(b) Performance victim VM *img-dnn*

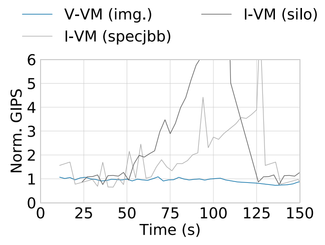


(c) GIPS victim VM *silo*

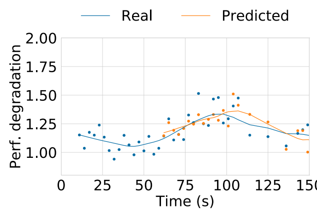


(d) Performance victim VM *silo*

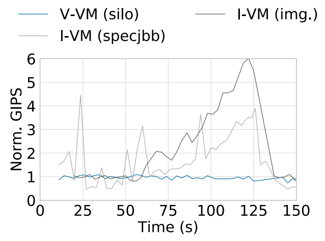
Fig. 11. Normalized GIPS and performance degradation (i.e., 95th tail latency) of experiments performed with 2 VMs executing *img-dnn* and *silo* as victim (V) and inflicting (I) VMs (plots a and b) and vice-versa (plots c and d).



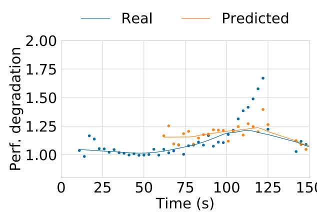
(a) GIPS victim VM *img-dnn*



(b) Performance victim VM *img-dnn*



(c) GIPS victim VM *silo*



(d) Performance victim VM *silo*

Fig. 12. Normalized GIPS and performance degradation (i.e., 95th tail latency) of experiments performed with 3 VMs executing *img-dnn*, *silo* and *specjbb* as victim (V) or inflicting (I) VMs. In plots a and b *img-dnn* is the victim and in plots c and d *silo* takes this role.

Additionally, Cloud White is able to determine that little or no performance degradation is taking place in the last third of the execution, when the co-running VM is no longer stressing the system main resources. This proves that Cloud White can be effectively used to detect performance degradation.

7.3. Prediction effectiveness with multiple latency-critical applications

This section evaluates how Cloud White behaves when running two and three VMs with latency-critical applications. For this purpose, we have chosen *img-dnn*, *silo* and *specjbb*. The first two applications were chosen since they exhibited high sensitivity (*img-dnn*, see Fig. 9(a)) and low sensitivity (*silo*, see Fig. 9(c)) to the interference. For the three VMs experiments, *specjbb* was added with an inflicting role since it presents a significant consumption of shared resources and performs a high QPS [37]. Thus, it introduces a high pressure and interference.

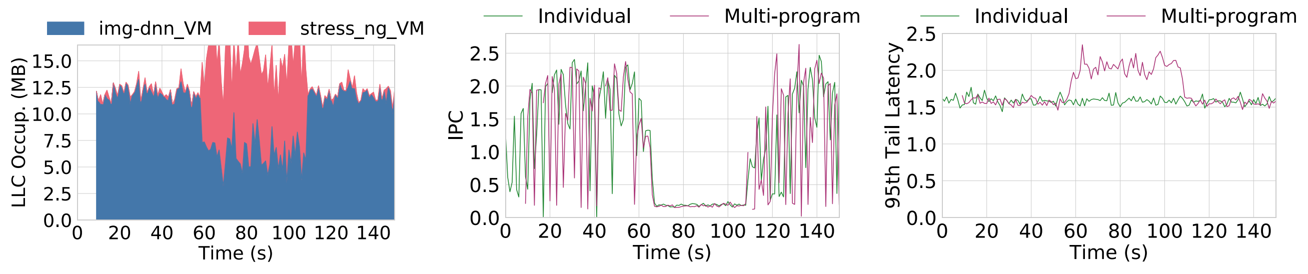
The main objective is to check Cloud White's two major design issues: (i) distinguish the victim VM and, (ii) accurately estimate its performance degradation. Although we have tested up to three VMs concurrently running latency-critical workloads, Cloud White is not limited to three. Notice, however, that in real public systems, not many latency-critical workloads are co-located in the same node due to their QoS requirements, so it is not realistic to test a high number of latency-critical workloads.

Two latency-critical applications. This section analyzes Cloud White's behavior when running two latency-critical applications. To this end, we followed the methodology (mapping of VMs to VCPUs and the way in which the interference is introduced) described in Section 7.1. Fig. 11 shows the results for the first 150 s of two experiments performed with the pair made by *img-dnn* and *silo*; in the first experiment, *img-dnn* acts as the victim (constant 50% CPU load) and *silo* as the inflicting VM, and in the second, both applications exchange the role (silo runs now with constant 30% CPU load). The two upper plots (Figs. 11(a) and 11(b)) show the results for the first experiment. Fig. 11(a) shows the normalized GIPS with respect to the steady scenario (first 50 s), before *silo* starts to increase its load. As already studied in Section 4, the GIPS of the victim VM (i.e., *img-dnn*) remain almost constant if the load experiences no variations, while the GIPS increase if the load increases, as it happens for the inflicting VM (i.e., *silo*). It can be observed that the load increase (from second 50 to second 110 approx.) matches the top of the curve of Fig. 11(b) on the right side, which draws the real and predicted performance degradation of *img-dnn*, the victim VM. Around second 60, Cloud White detects that the load of the co-running VM has increased significantly and starts to estimate the effect it has on *img-dnn*'s performance. Notice that after second 110 approx., the load of the inflicting VM reduces, and so does the interference it introduces, until it reaches again the steady load. In this experiment, Cloud White is able to estimate the performance degradation with an overall prediction error by 6% and average dynamic prediction error of less than 10%. A similar reasoning can be applied to the two bottom figures (Figs. 11(c) and 11(d)), which correspond to the second experiment where *img-dnn* is the inflicting application. In this case, the average dynamic prediction error is roughly the same but the overall prediction error slightly lowers to 4%.

Three latency-critical applications. Finally, we analyze how Cloud White behaves with three co-located VMs running latency-critical workloads. For illustrative purposes, Fig. 12 presents the results for *img-dnn*, *silo* and *specjbb*. In the first experiment, *img-dnn* is the victim application (constant 50% CPU load) and the other applications act as inflicting (Figs. 12(a) and 12(b)). In the second experiment, *silo* acts as the victim application with a constant 20% CPU load (Figs. 12(c) and 12(d)). As in the previous section, the normalized GIPS of the victim VM remain around 1 for the whole execution, which allows Cloud White to identify it; meanwhile the inflicting VMs show an increasing trend of the GIPS, matching the load increase. For the steady-load victim VM, Cloud White is able to accurately estimate the performance degradation in both cases, with an overall prediction error by 7.5% in the case where *img-dnn* is the victim VM and 8% in the case where *silo* is the victim VM and an average dynamic prediction error of 11% and 8.5%, respectively.

8. Comparison to prior work

Cloud White is able to detect the inter-VM interference (i) considering VMs as "black boxes" while (ii) assuming these VMs run



(a) LLC occupancy in multi-program (b) Performance of *stress_ng* in isolation (c) Performance of *img-dnn* in isolation

Fig. 13. Comparison of the LLC occupancy and performance of *img-dnn* and *stress_ng*.

latency-critical applications. Combining both statements together is one of the main contributions of this work.

This section compares our approach with the state-of-the-art approach Alita [14]. For this purpose, we implemented Alita in our experimental platform. As our proposal, this approach also handles VMs as black boxes; hence, the comparison concentrates on analyzing the online interference detection capabilities and differences between both approaches.

Alita uses low-level information although different from that used by Cloud White. The major difference is not only in the metrics used but how the interference is detected. On the one hand, Cloud White combines numerous hardware events in a multivariable regression model that estimates performance degradation considering interactions between resources. Alita simplifies the process estimating degradation in three main shared resources (LLC, memory bus and CPU) in an independent way. Next, we compare the effectiveness of Cloud White against Alita from the point of view of latency-critical applications.

LLC contention. Alita evaluates LLC contention based on the *fair LLC quota* each VM should occupy, which is determined by the number of CPUs assigned to the VM. Nonetheless, as proved in [47], LLC occupancy is not a good metric to quantify contention since it does not consider data reuse.

Fig. 13 presents a counter example to illustrate this claim, where *img-dnn* is executed together with the stressor microbenchmark (*stress_ng*), each on a VM with 8 VCPUs. *Img-dnn* shows the same behavior across all the execution while the microbenchmark occupies little LLC space (less than 1 MB) except in the second third of the execution (from 60 s to 110 s) where it stresses the LLC, polluting it with data with little locality (i.e., no reuse). In the first and last third of the execution, *img-dnn* occupies most of the LLC space (over 10 MB out of 16.5 MB). If computed the fair LLC quota, each VM should occupy 5.5MB (8/24 × 16.5). Thus, Alita detects *img-dnn* is polluting the LLC when, in fact, it is not since the IPC of *stress_ng* (see Fig. 13(b)) is almost the same as that achieved when running alone. On the contrary, in the second third of the execution Alita does not detect the real LLC contention (see *img-dnn*'s tail latency increase in Fig. 13(c)) since both VMs have a balanced LLC consumption. As opposed to Alita, Cloud White would properly detect this interference since (i) *img-dnn* would not be detected as inflicting as its GIPS do not vary, and (ii) *stress_ng* would be detected as inflicting in the middle phase where the GIPS increase.

Memory Bus contention. Alita quantifies memory bus contention by detecting *split locks* [14], which deeply degrade the performance of latency-critical applications. This approach works well on specific workloads like malicious tenant programs. We do not observe, however, split locks in the studied Tailbench applications. Moreover, when memory bandwidth contention appears in regular workloads it is not detected by Alita unlike Cloud White which considers memory-related metrics.

CPU contention. Contention in the CPU is quantified in terms of power. In the case of CPU-intensive applications, Alita detects CPU contention since these applications achieve a high IPC and present high circuitry activity, which significantly increases the dynamic power and temperature. On the contrary, latency-critical applications do not stress so much the CPU and thus, we observed little increase in temperature (a few degrees Celsius at most) when running such applications.

In summary, Alita is suited to work well with specific workloads presenting a high interference but cannot be considered as a general approach. In contrast Cloud White is, to the best of our knowledge, the only approach able to detect *smooth* interference caused by inflicting VMs running latency-critical applications.

9. Applying Cloud White to improve QoS

The objective of Cloud White is to detect the inter-VM interference and quantify the performance degradation suffered by the victim VM(s). This information can be leveraged by cloud providers to carry out specific actions depending on the level of performance degradation. For instance, urgent solutions need to be carried out where there exists a narrow margin with SLA violations. The most straightforward action that the cloud provider can carry out on such a case is to move the identified victim VM to another less-loaded node. Notice that if this VM is not properly identified, the cloud provider has no way of knowing which VM to move. Other more refined actions for not so urgent situations may be more appropriate than migration since this process is resource-intensive [6,48], and for VMs with a large image size, it can be very costly. These actions include analyzing the major shared processor resource that bottlenecks the performance due to interference. For instance, in case it is the LLC or memory bandwidth the resource that experiences a higher utilization increase, then the cloud provider can assign a larger share of it (e.g., LLC cache ways or memory bandwidth) to the victim VM in order to rise its performance to the same level it was before the inflicting action.

10. Conclusions

Accurately estimating performance degradation of latency-critical workloads is a key challenge for cloud providers since it allows them to improve resource utilization while meeting applications QoS requirements. This work has presented an in-depth analysis of how performance degradation can be identified and quantified in a realistic scenario with multiple co-located VMs, handling VMs as black boxes and relying on metrics that can be easily monitored in the public cloud.

The proposed approach, Cloud White, uses the GIPS metric as a novel way to discern victim VM(s), which have a steady load over the last quanta but their performance degrades due to the interference introduced by the inflicting VM(s). After that, it estimates the performance degradation of the victim VM(s) using multi-variable linear regression models.

Results show that Cloud White is able to accurately identify the victim VM(s) in situations of multiple VMs running together. Moreover, we have shown that Cloud White is able to do that dynamically by detecting the co-running VMs that experience load changes. Upon a load change, Cloud White estimates the performance degradation of the victim VMs in an accurate way. Experimental results show that the total error deviation is about 5% and the average dynamic prediction error less than 10%. To the best of our knowledge, this is the first approach that detects and quantifies inter-VM interference in cloud systems running latency-critical applications.

CRediT authorship contribution statement

Lucía Pons: Software, Investigation, Data curation, Visualization, Writing – original draft, Writing – review & editing. **Josué Feliu:** Software, Investigation, Writing – original draft, Formal analysis, Writing – review & editing. **Julio Sahuquillo:** Conceptualization, Project administration, Funding acquisition, Supervision, Writing – review & editing. **María E. Gómez:** Resources, Writing – review & editing. **Salvador Petit:** Conceptualization, Methodology, Formal analysis, Writing – review & editing. **Julio Pons:** Resources, Visualization, Software. **Chaoyi Huang:** Supervision, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Julio Sahuquillo reports financial support was provided by Huawei Cloud.

Acknowledgments

This work has been supported by Huawei Cloud, and in part by Spanish Ministerio de Universidades under grant FPU18/01948, and by Spanish Ministerio de Universidades and European ERDF under grants RTI2018-098156-B-C51 and PID2021-123627OB-C51. Funding for open access charge: CRUE-Universitat Politécnica de Valéncia.

References

[1] Amazon's EC2 [online], 2020, Available at http://aws.amazon.com/ec2/, Accessed: 2020-09-30.
[2] Windows azure [online], 2020, Available at http://www.windowsazure.com/, Accessed: 2020-09-30.
[3] Google compute engine [online], 2020, Available at https://developers.google.com/compute/, Accessed: 2020-09-30.
[4] J.R. Hamilton, Cost of power in large-scale data centers, 2020, Available at https://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/, Accesed: 2020-09-27.
[5] C. Lu, K. Ye, G. Xu, C.-Z. Xu, T. Bai, Imbalance in the cloud: An analysis on alibaba cluster trace, in: 2017 IEEE International Conference on Big Data, 2017, pp. 2884–2892, http://dx.doi.org/10.1109/BigData.2017.8258257.
[6] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, R. Bianchini, Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms, in: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), 2017, pp. 153–167, http://dx.doi.org/10.1145/3132747.3132772.
[7] Q. Liu, Z. Yu, The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from alibaba trace, in: Proceedings of the ACM Symposium on Cloud Computing (SoCC), 2018, pp. 347–360, http://dx.doi.org/10.1145/3267809.3267830.
[8] H. Yang, A. Breslow, J. Mars, L. Tang, Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers, in: Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA), 2013, pp. 607–618.
[9] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for QoS-aware clouds, in: Proceedings of the 5th European Conference on Computer Systems (EuroSys), 2010, pp. 237–250.
[10] S. Shekhar, H. Abdel-Aziz, A. Bhattacharjee, A. Gokhale, X. Koutsoukos, Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications, in: Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 82–89, http://dx.doi.org/10.1109/CLOUD.2018.00018.
[11] R. Nishtala, V. Petrucci, P. Carpenter, M. Sjalander, Twig: Multi-agent task management for colocated latency-critical cloud services, in: Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 167–179, http://dx.doi.org/10.1109/HPCA47549.2020.00023.
[12] T. Patel, D. Tiwari, CLITE: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers, in: Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 193–206, http://dx.doi.org/10.1109/HPCA47549.2020.00025.
[13] D. Novaković, N. Vasić, S. Novaković, D. Kostić, R. Bianchini, DeepDive: Transparently identifying and managing performance interference in virtualized environments, in: Proceedings of USENIX Annual Technical Conference (ATC), 2013, pp. 219–230.
[14] Q. Chen, S. Xue, S. Zhao, S. Chen, Y. Wu, Y. Xu, Z. Song, T. Ma, Y. Yang, M. Guo, Alita: Comprehensive performance isolation through bias resource management for public clouds, in: Proceedings of SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, 2020, pp. 1–13, http://dx.doi.org/10.1109/SC41405.2020.00036.
[15] M. Melo Alves, L.M.d.A. Drummond, A multivariate and quantitative model for predicting cross-application interference in virtual environments, J. Syst. Softw. 128 (C) (2017) 150–163, http://dx.doi.org/10.1016/j.jss.2017.04.001.
[16] S.A. Javadi, A. Suresh, M. Wajahat, A. Gandhi, Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments, in: Proceedings of the ACM Symposium on Cloud Computing (SoCC), 2019, pp. 272–285, http://dx.doi.org/10.1145/3357223.3362734.
[17] N. Vasic, D.M. Novakovic, S. Miucin, D. Kostic, R. Bianchini, DejaVu: accelerating resource allocation in virtualized environments, in: Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012, pp. 423–436.
[18] S. Chen, C. Delimitrou, J.F. Martínez, PARTIES: QoS-aware resource partitioning for multiple interactive services, in: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019, pp. 107–120, http://dx.doi.org/10.1145/3297858.3304005.
[19] R. Nishtala, P. Carpenter, V. Petrucci, X. Martorell, Hipster: Hybrid task manager for latency-critical cloud workloads, in: Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 409–420, http://dx.doi.org/10.1109/HPCA.2017.13.
[20] C. Delimitrou, C. Kozyrakis, Quasar: Resource-efficient and QoS-aware cluster management, SIGPLAN Not. 49 (4) (2014) 127–144, http://dx.doi.org/10.1145/2644865.2541941.
[21] H. Moradi, W. Wang, D. Zhu, Online performance modeling and prediction for single-VM applications in multi-tenant clouds, IEEE Trans. Cloud Comput. (2021) http://dx.doi.org/10.1109/TCC.2021.3078690.
[22] Intel Cloud Technology, Are Noisy Neighbors in Your Data Center Keeping You Up at Night? [online], Tech. Rep., Intel Corporation, 2017.
[23] T. Gleixner, I. Molnar, Performance counters for linux, 2009.
[24] P. Irelan, S. Kuo, Performance Monitoring Unit Sharing Guide, Tech. Rep., Intel Corporation, 2009.
[25] Intel Corporation, Intel RDT library, 2019, https://github.com/intel/intel-cmt-cat/tree/master/lib.
[26] Red Hat, Libvirt virtualization API [online], 2020, Available at http://libvirt.org, Accessed: 2020-09-30.
[27] Open vSwitch [online], 2020, Available at https://www.openvswitch.org/, Accessed: 2020-09-28.
[28] O. Sefraoui, M. Aissaoui, M. Eleuldj, OpenStack: Toward an open-source solution for cloud computing, Int. J. Comput. Appl. 55 (3) (2012) 38–42.
[29] API overview - elastic cloud server [online], 2022, https://support.huaweicloud.com/intl/en-us/api-ecs/ecs_01_0008.html, Accessed: 2022-05-16.
[30] Choosing a hypervisor - arch-design documentation openstack.org [online], 2022, https://docs.openstack.org/arch-design/design-compute/design-compute-hypervisor.html, Accessed: 2022-05-16.
[31] DPDK [online], 2020, https://www.dpdk.org/, Accessed: 2020-09-28.
[32] S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, Ceph: A Scalable, High-Performance Distributed File System, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), 2006, pp. 307–320.

[33] Kunpeng BoostKit for Virtualization, Tech. Rep. Issue 11, Huawei Technologies Co., Ltd., 2021, URL https://support.huaweicloud.com/intl/en-us/twp-kunpengcpfs/kunpengcpfs-twp.pdf.
[34] H. Kasture, D. Sanchez, Tailbench: a benchmark suite and evaluation methodology for latency-critical applications, in: Proceedings of the 12th IEEE International Symposium on Workload Characterization (IISWC), 2016, pp. 1–10.
[35] R. Baeza-Yates, Applications of web query mining, in: European Conference on Information Retrieval, 2005, pp. 7–22.
[36] D.G. Feitelson, Workload Modeling for Computer Systems Performance Evaluation, Cambridge University Press, 2015.
[37] L.a. Pons, J. Feliu, J. Puche, C. Huang, S. Petit, J. Pons, M.a.E. Gómez, J. Sahuquillo, Effect of hyper-threading in latency-critical multithreaded cloud applications and utilization analysis of the major system resources, Future Gener. Comput. Syst. 131 (C) (2022) 194–208, http://dx.doi.org/10.1016/j.future.2022.01.025.
[38] Canonical Ltd, Ubuntu manpage: stress-ng, 2020, https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html, Accessed: 2020-09-30.
[39] ESnet, NLANR, DAST, Iperf tool for network bandwidth measurements, 2020, https://iperf.fr/, Accessed: 2020-09-30.
[40] J. Li, N.K. Sharma, D.R.K. Ports, S.D. Gribble, Tales of the tail: Hardware, OS, and application-level sources of tail latency, in: Proceedings of the ACM Symposium on Cloud Computing (SoCC), 2014, pp. 1–14, http://dx.doi.org/10.1145/2670979.2670988.
[41] S. Seabold, J. Perktold, statsmodels: Econometric and statistical modeling with python, in: Proceedings of 9th Python in Science Conference, 2010, pp. 92–96.
[42] J. Wakefield, Non-linear regression modelling and inference, in: Methods and Models in Statistics: In Honour of Professor John Nelder, FRS, World Scientific, 2004, pp. 119–153.
[43] S. Dodd, A. Bassi, K. Bodger, P. Williamson, A comparison of multivariable regression models to analyse cost data, J. Eval. Clin. Pract. 12 (1) (2006) 76–86.
[44] M. Valbuena, D. Sarabia, C. de Prada, A reduced-order approach of distributed parameter models using proper orthogonal decomposition, in: Proceedings of 21st European Symposium on Computer Aided Process Engineering, Vol. 29, 2011, pp. 26–30, http://dx.doi.org/10.1016/B978-0-444-53711-9.50006-7.
[45] E. López-González, M. Ruiz-Soler, Análisis de datos con el modelo lineal generalizado. Una aplicación con R, Rev. Esp. Pedagog. (2011) 59–80.
[46] J.D. Triveri, LOESS - nonparametric scatterplot smoothing in Python, 2018, http://www.jtrive.com/loess-nonparametric-scatterplot-smoothing-in-python.html, Accessed: 2020-12-21.
[47] L. Pons, J. Sahuquillo, V. Selfa, S. Petit, J. Pons, Phase-aware cache partitioning to target both turnaround time and system performance, IEEE Trans. Parallel Distrib. Syst. 31 (11) (2020) 2556–2568, http://dx.doi.org/10.1109/TPDS.2020.2996031.
[48] R.W. Ahmad, A. Gani, S.H.A. Hamid, M. Shiraz, A. Yousafzai, F. Xia, A survey on virtual machine migration and server consolidation frameworks for cloud data centers, J. Netw. Comput. Appl. 52 (2015) 11–25, http://dx.doi.org/10.1016/j.jnca.2015.02.002.

Lucía Pons received the BS and MS degrees in computer engineering from the Universitat Politècnica de València (UPV), Spain, in 2018 and 2019, respectively. She is currently working towards a Ph.D. degree at the Department of Computer Engineering (DISCA) of the same university. Her Ph.D. research focuses on cache partitioning approaches and efficient use of shared resources in multi-core.

Josué Feliu received his M.Sc. and Ph.D. degrees in computer engineering from the UPV, Spain, in 2012 and 2017, respectively. He is currently working as a postdoctoral researcher at the Universidad de Murcia. His research interests include scheduling strategies and performance modeling for multicore and multi-threaded processors. He was awarded the "IEEE TCSC Outstanding Ph.D Dissertation Award" in 2017.

Julio Sahuquillo received the BS, MS, and Ph.D. degrees from the UPV, Spain, all in computer engineering. He is a Full Professor with the DISCA department at the UPV. He has taught several courses on computer organization and architecture. He has authored over 150 refereed conference and journal papers. His current research interests include multicore processors, memory hierarchy design, GPU architecture, and resource management.

María E. Gómez received his BS, MS, and Ph.D. degrees in Computer Engineering from the UPV, Spain, in 1996 and 2000, respectively. She joined the Department of Computer Engineering (DISCA) at Universitat Politècnica de València in 1996 where she is currently a Full Professor. She has published more than 80 conference and journal papers. She has served on program committees for several major conferences. Her research interests are on processor architecture and interconnection networks.

Salvador Petit received the Ph.D. degree in computer engineering from the UPV. Since 2009, he has been an Associate Professor with the DISCA department, where he has taught several courses on computer organization. He has authored over 100 refereed conference and journal papers. His current research interests include multi-core processors, memory hierarchy design, GPU architecture, and resource management.

Julio Pons received the BS, MS, and Ph.D. degrees from the UPV, Spain, all in computer engineering. He is an Associate Professor with the DISCA department. He has taught several courses on computer organization and operating systems. His current research interests include multi-core processors, memory hierarchy design, cache sharing and cloud computing.

Chaoyi Huang received his BS degree in mechanical engineering, MS degree in computer aided quality management from Huazhong University of Science \& Technology, China. He is an expert in Cloud BU, Huawei Technologies Co., Ltd. His current research interests is improving cloud resource efficiency, including technologies like inter-VM interference detection and control, dynamic VM resizing.